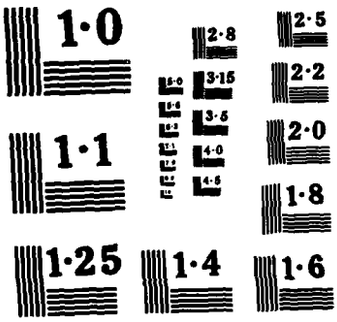
AD-A157 764 SOME EXPERIMENTS IN VLSI LEAF-CELL OPTIMIZATION(U) 1/1
PRINCETON UNIV NJ DEPT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE K IWANO ET AL. 1984 ARO-18985.25-EL
UNCLASSIFIED DAAG29-82-K-0095 F/G 9/5 NL

END
FILMED
DTIC



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(2)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARO 18985.25-EL	2. GOVT ACCESSION NO. N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) Some Experiments in VLSI Leaf-Cell Optimization		5. TYPE OF REPORT & PERIOD COVERED Reprint 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kazuo Iwano, Kenneth Steiglitz		8. CONTRACT OR GRANT NUMBER(s) DAAG29-82-K-0095
9. PERFORMING ORGANIZATION NAME AND ADDRESS Princeton University		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Very Large Scale Integration
Arrays
Cells
Optimization
Logic Circuits
Delay

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1984 IEEE Workshop on VLSI Signal Processing
University of Southern California
Nov. 12-14, 1984

Some Experiments in VLSI Leaf-cell Optimization†

Kazuo Iwano

Kenneth Steiglitz

Department of Electrical Engineering and Computer Science
Princeton University
Princeton, N. J. 08544

Abstract

This paper describes a method for local optimization of VLSI leaf cells, using the parameterized procedural layout language ALLENDE [5]. Tradeoffs among delay time, power consumption, and area are illustrated. Three different implementations of the 1-bit full adder are compared: a random logic circuit, a data selector, and a PLA. The fastest random logic 1-bit full adder has a time-power product about 1/3 that of the fastest data selector, and about 1/4 that of the fastest PLA. The 4-bit parallel adder is used to illustrate the effect of loading when leaf cells are combined.

1. Introduction

In the design of a custom VLSI chips it often happens that there is one cell that is used many times, usually in an array or a recursive structure. The fact that a cell is used many times means that there is a large potential payoff in its optimization, and that the problem can be made small enough to be manageable. Arrays of cells are especially common in digital signal processing applications, where regular structures, like systolic arrays, lead to designs that are easy to lay out efficiently, and have high throughput. As examples, bit-parallel and bit-serial multipliers can be constructed from one- and two-dimensional arrays of one-bit full adders, as can a wide variety of pipelined FIR and IIR filters (see [1], for example). As another example, a processor for updating one-dimensional cellular automata has been designed at Princeton which consists of a one-dimensional array of 5-input/1-output PLA's [10]. In such cases the problem of making most efficient use of a given piece of silicon breaks down into two distinct problems: 1) choice of the global packing strategy (the method of laying out and interconnecting leaf cells, and connecting them to power and clocks), and 2) the design of the iterated structure itself (which we call the *leaf cell*). In this paper we study the second problem: the design of efficient leaf cells. The example used throughout is the most common in digital signal processing, the 1-bit full adder.

There are three important measures of how good a leaf cell is: its time delay T; its peak or average power dissipation P_{max} or P_{ave}; and its area A.

† This work was supported by National Science Foundation Grants ECS-8307955, U.S. Army Army Research Office, Durham, NC, under Grant DAAG29-82-K-0095, DARPA Contract N00014-82-K-0549, and ONR Grant N00014-83-K-0275.

Ideally, the designer should be able to trade off these measures, one against the other. For example, in one application the clock may be fixed at a known value T_0, and it would therefore be senseless to make the the cell faster. On the other hand, peak power may be a real constraint because of heat dissipation limitations, and at the same time it may be important to keep the area small so as to fit as many cells on one chip as possible. We might therefore try to minimize some measure of the peak power and area (the product, for example), while enforcing the constraint $T \leq T_0$. In other applications speed may be critical, and it may be important to minimize T while observing constraints on P_p and A, and so on. In general, we would like to have enough information about the tradeoffs among the measures T, P and A to make intelligent design decisions. As we will see, the $P-T$ tradeoff is often of most interest, since the area is often a less sensitive function of design parameters (at least for fixed topology).

2. Formulation

The basic approach we take will be to search for local improvements on random initial designs. The search strategy will be to consider all single or double changes in element size along the critical path. When only single changes are tried, we call the procedure "1-change", when double changes are tried, "2-change". The idea is that the critical path indicates which parameters are most important to performance at any given point in the analysis.

We will limit the optimization to choice of pulldown widths. The method can be extended to choice of layers, orientation, and topologies. We will, however, study three radically different topologies for the full adder: the PLA, data-selector, and random logic.

The main analysis tools used in these experiments are the timing simulator CRYSTAL, and the power-estimation program POWEST, together with the rest of the Berkeley tool package [2].

Another essential component of the work is a procedural, constraint-based layout language for specifying VLSI layouts; in this case, we used the new language ALLENDE being developed at Princeton, a successor to ALI2 and CLAY [3,4,5]. This allows us to specify circuit parameters and have a cifplot generated automatically.

3. The Critical-Path Optimization

Figure 1 shows how the optimization is performed in our experiments. In Figure 1 **faparm** is an input parameter vector to **ANALYSIS** which has diffusion widths of nodes as described in section 4. The initial **faparm** is generated at random by **RANDOM** according to its input file **pattern**. **ANALYSIS** takes **faparm** as its input and generates an appropriate layout and its resulting **T**, **P**, and **A**, as well as **the nodes on the critical path** (hereafter called the critical path nodes). Since every node on the critical path has an associated parameter in **faparm**, **CASEGEN** can generate **faparms** as subcases by using the one-(two-)change method. Here the one-(two-)change method changes one(two) parameter(s) associated with the critical path nodes by one step. (From here on the 1-change method is denoted by 1-opt or Random 1-opt, and the 2-change method by 2-opt or Random 2-opt.)

The optimization strategy is shown in the flowchart of figure 2. When the first improvement occurs, this case is picked up for the next iteration. If no improvement occurs but there exists a case which has the same cost and has

not yet been analyzed, this case is adopted next. Otherwise a new random **faparm** is generated for the next iteration, to search for other locally optimal points. We used two cost criteria for optimization: T, and $P_{max}T$ (hereafter denoted by PT). Figure 3 shows an outline of the main procedures used in the **ANALYSIS** loop. A short description of each follows below:

1) **ALLENDE** — This procedural constraint-based VLSI layout language produces an integrated circuit layout in Caltech International Form (CIF) corresponding to the specified parameters [5].

2) **MEXTRA** — MEXTRA reads CIF and extracts the nodes to create a circuit description for further analyses [2].

3) **CRYSTAL** — CRYSTAL is used for finding the worst-case delay time of the circuit [2].

4) **POWEST** — POWEST is used for finding the average and maximum power consumption of the circuit.

5) **CRITICAL** — CRITICAL reports the critical path nodes by using the output of CRYSTAL.

6) **LIST** — This command stores the vector of results (**T,P,A**) in the HISTORY file for further optimization.

In figure 3 the squares surrounded by dotted lines are files used for inputs or outputs of the above procedures.

1) **faparm** — The faparm has parameters for layout generation; for example, the diffusion width of each node, the permutation of product terms in a PLA, etc.

2) **layout generating program** — There are several ALLENDE programs implementing desired circuit topologies such as the PLA, random logic, etc. Each program requires parameters in its corresponding faparm.

3) **the critical path nodes** — The critical path nodes are extracted from the output of CRYSTAL. Each node can be associated with parameters in faparm. This is done by looking up a table for each topology, which associates each node with its corresponding parameter.

4. Full-Adder Circuit Implementations

As mentioned in the Introduction, we adopted the 1-bit full-adder circuit as an example for experimentation, because it is relatively simple, but is a basic arithmetic logic circuit. The 1-bit full-adder circuit can be implemented in many ways. We chose three kinds of circuits: the **PLA**, **Data Selector**, and **Random logic**. Each layout has several parameters. We will use the vector representation of these parameters; that is $d = (d_1, d_2, \ldots, d_n)$ means that the diffusion width of node i is $d_i\lambda$. We also use the vector $k = (k_1, k_2, \ldots, k_n)$ to mean that the pullup to pulldown ratio of the inverter, NOR, or NAND circuit in which node i exists is k_i. The vector k is fixed for each circuit.

1) PLA

Figure 4 shows the full-adder circuit diagram implemented by a programmable logic array (PLA) [7]. This layout has the following 17 parameters and 2 permutations.

$d = (d_{and_1}, \ldots, d_{and_7}, d_{or_1}, d_{or_2}, d_{in_{1,1}}, \ldots, d_{in_{3,2}}, d_{out_1}, d_{out_2}, \pi_1, \pi_2)$

$k = (4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)$

- 7 pulldown diffusion widths of the AND plane.
- 2 pulldown diffusion widths of the OR plane.
- 6 pulldown diffusion widths for inputs.
- 2 pulldown diffusion widths for outputs.
- 1 permutation of product terms in the AND plane.
- 1 permutation of outputs.

In the optimization process, the two permutations are fixed for the sake of simplicity. However those two permutations are chosen in advance in order to give the best result before the optimization by doing experiments based on various random permutations as inputs.

2) **Berkeley PLA**

The PLA generated by using **mkpla** of the Berkeley VLSI tools [2,8] is used for the purpose of cost comparison with the PLA implemented in 1). This PLA is not optimized, but uses the following fixed parameter vector.

$d = (4,4,4,4,4,4,4,4,4,8,8,8,8,8,8,8,8)$

$k = (4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)$

3) **Data Selector**

Figure 5 shows the full-adder circuit diagram of a Data Selector implementation [9]. The following truth table is used.

C_i	B	S	C_o
0	0	A	C_i (or B)
0	1	$\bar{A}$	A
1	0	$\bar{A}$	A
1	1	A	C_i (or B)

This circuit selects inputs (A, $\bar{A}$, or C_i) instead of calculating S and C_o. Here C_i is the input carry signal, C_o is the output carry signal, and S is the output sum signal. A and B denote the two other inputs. This layout has the following 8 parameters.

$d = (d_A, d_B, d_{C_i}, d_1, d_2, d_3, d_{C_o}, d_S)$

$k = (4,4,4,4,8,4,8,8)$

- 3 pulldown diffusion widths for input inverters.
- 3 pulldown diffusion widths for internal inverters.
- 2 pulldown diffusion widths for output inverters.

4) **Random Logic**

Figure 6 shows the circuit diagram of the Random Logic Implementation [6].

This layout has the following 4 parameters.

$d = (d_1, d_2, d_{C_0}, d_S)$

$k = (8, 12, 4, 4)$

- 2 pulldown diffusion widths for internal inverters.
- 2 pulldown diffusion widths for output inverters.

All the circuits above were verified by **ESIM** [2] or **SIMULATE** [5].

5. Parameterization

The diffusion width of the pullup in each stage is automatically determined and implemented by ALLENDE in the following way. Suppose that the current parameter vector is $d = (d_1, d_2, \ldots, d_n)$, and the pullup-to-pulldown ratio vector of the specified layout is $k = (k_1, k_2, \ldots, k_n)$. (The choice of pullup-to-pulldown ratio is discussed in [7].) For each node i, define the variables Z_{pu}, Z_{pd}, and a pullup-to-pulldown ratio K as follows.

$$Z_{pu} = \frac{L_{pu}}{W_{pu}}, \quad Z_{pd} = \frac{L_{pd}}{W_{pd}}, \quad K = \frac{Z_{pu}}{Z_{pd}}$$

where
L_{pu} (L_{pd}) is the length of pullup (pulldown).
W_{pu} (W_{pd}) is the width of pullup (pulldown).
$W_{pd} = d_i$, $K = k_i$ and $L_{pd} = 2$.

L_{pu} and W_{pu} are determined as follows.

If $W_{pd} \leq 2K$

$$W_{pu} = 2$$
$$K = \frac{L_{pu}/2}{2/W_{pd}} \quad \text{or} \quad L_{pu} = \frac{4K}{W_{pd}}.$$

If $W_{pd} > 2K$

$$W_{pu} = W_{pd} / K$$
$$K = \frac{L_{pu}/W_{pu}}{2/W_{pd}} \quad \text{or} \quad L_{pu} = \frac{2KW_{pu}}{W_{pd}}.$$

We adopted following choices.

1) $\lambda = 2\mu$
2) The timing estimation program CRYSTAL uses an input pulse which is 1 *nsec* wide.

6. Results

Table 1 shows a comparison of the performance of our implementations. Each row represents one locally optimal point using as criterion the item indicated by *. The units of A, P_{ave}, P_{max}, T, APT and PT are λ^2, $(10^{-6} * W)$, $(10^{-6} * W)$, ns, $(10^{-12} * \lambda^2 * W * ns)$ and $(10^{-6} * W * ns)$ respectively in all tables. Figure 7 shows P_{max} vs T curves for different topologies, while figure 8 shows several P_{max} vs T trajectories obtained during the process of optimization using the 1-change and 2-change methods for the Data Selector and the Random

Table 1. performance comparison (1 bit full adder)

type	A	P_{ave}	P_{max}	T	APT	PT	parameter
PLA	21560	6472	10183	12.8*	2802	1303	1)
	21840	5678	9241	15.3*	3087	1413	2)
	21762	5503	8616	14.9*	2794	1284	3)
PLA(Berkeley)	22176	7314	11749	12.8*	3339	1504	4)
Data Selector	8100	3765	6117	15.8*	783	966	8 8 8 8 8 8 8 8
	8100	3529	5645	16.5*	754	931	8 8 8 4 8 8 8 8
	8190	3764	6116	15.9*	796	972	12 8 8 8 8 8 8 8
Random Logic	7742	1331	1957	16.5*	392	323	16 12 3 2
	9600	1683	2427	16.4*	382	398	16 24 2 3
	9800	1644	2329	16.4*	378	382	16 24 2 2
	9600	1723	2506	16.5*	397	413	16 24 3 3
	5194	705	1096	22.6	128	248*	6 8 2 2
	4704	626	1018	25.9	124	264*	4 6 3 2
	5136	744	1174	22.9	138	269*	6 8 2 3

1) d = (4,4,4,4,4,4,4,3,4,4,8,8,8,4,4,4,8,2)

2) d = (4,2,3,3,3,3,3,3,4,3,8,8,8,4,4,4,8,2)

3) d = (3,3,3,4,4,4,4,4,3,3,8,8,8,4,4,4,4,3)

4) d = (4,4,4,4,4,4,4,4,4,4,8,8,8,8,8,8,8,8)

Table 2 performance comparison (4 bit parallel adder)

type	A	P_{ave}	P_{max}	T	APT	PT	parameter
Data Selector	41310	16536	28218	75.3*	877761	212482	4 8 8 8 16 8 16 16
	44550	16536	28218	84.1*	1057230	237313	4 8 8 8 16 8 24 16
	45409	16534	28213	84.3*	1079990	287836	4 8 8 16 16 16 16 16
	44523	13248	21641	91.0*	876805	196933	4 8 8 8 16 4 8 4
	42845	12301	19748	92.5*	782645	182669	4 8 4 4 16 8 8 4
	43747	11362	17868	94.9*	741806	169567	4 8 4 4 16 4 8 4
	43605	12354	20692	98.0*	884229	202782	2 8 4 8 16 8 4 8
	45441	11885	19753	100.8*	904777	199110	2 8 4 8 16 8 4 4
	44523	12305	19755	101.1*	889227	199723	4 8 4 8 16 4 4 8
	44849	11831	18808	103.2*	866631	194099	4 8 4 4 16 8 4 4
	43747	11362	17868	103.8*	809812	185112	4 8 4 4 16 4 4 8
Random Logic	35552	6577	10335	41.1*	151014	42476	16 12 8 2
	34848	6734	10649	41.4*	153634	44087	16 12 8 3

Logic circuit. Each point takes about 1.5 minutes of cpu time on a VAX 11/750. Many of the locally optimal solutions have identical parameter values on the critical path, but differ in other coordinates because of different random starting values.

7. Parallel Adder : The effect of loading factors

The preceding results did not take the loading on the output of the circuit into account. When these circuits are used in arrays, this may become important. To study this problem, we implemented two circuits for a 4-bit parallel adder, using the **Data Selector** and the **Random Logic** 1-bit full adders of the previous section. The results are shown in Table 2.

8. Discussion of Results

8.1. P_{max} vs T tradeoff

Figure 8 shows P_{max}-T trajectories followed by the critical path optimization process, when minimizing T for the **Random Logic** circuit. The dotted envelope shows the final tradeoff curve for P vs T. Notice that the locally optimal point obtained by using PT as the cost criterion lies very close to the trajectory obtained when minimizing T. (See point **a**, with $P = 12.5mW$, and $T = 22.4ns$.) For comparison, the optimization for PT gave us a locally optimal point **b** with $P = 10.9mW$ and $T = 22.6ns$, very close to point **a**. Thus, optimization using the two criteria is consistent.

8.2. Performance comparison among the PLA, Data selector, and Random logic.

Table 3 normalized performance comparison (1-bit full adder)

type	A	P_{ave}	P_{max}	T	APT	PT
Random Logic	100	100	100	100	100	100
Data Selector	105	283	313	96	200	299
PLA	278	486	520	78	715	403
PLA(Berkeley)	286	550	600	78	852	466

Table 3 shows a normalized performance comparison of the best locally optimal point for each layout, minimizing T. The **Random Logic** seems to be the best choice in all respects except T. However, it is the fastest among the 4-bit parallel adder implementations. The T of the 4-bit parallel adder using **Random Logic** is less than 4 times the T of the 1-bit full adder, while in the other layouts it is more than 4 times the T of the 1-bit full adder. The reason is that this **Random Logic** 1-bit full adder circuit calculates the carry signal and propagates it before the calculation of the sum signal, so the carry ripple propagates faster than the sum. As a result, the 4-bit parallel adder takes only 2.5 times as much time as the 1-bit full adder. Figure 7 shows the P-T tradeoff curve of each layout. The curve for the **Random Logic** circuit is below the one for the **Data Selector**, which is below that for the **PLA**. Hence we can order the layouts with **Random logic** best, **Data Selector** next, and **PLA** last. This result agrees with our intuition because this order is the same as the order of circuit specialization.

8.3. Comparison between our PLA and the Berkeley PLA

Both **PLA**'s have almost the same costs, except for P. The reason is that our locally optimal point occurs at the choice d = (4,4,4,4,4,4,3,4,4,8,8,8,4,4,4,8,2),

while the **Berkeley PLA** adopts d = (4,4,4,4,4,4,4,4,4,8,8,8,8,8,8,8,8). The **Berkeley PLA** is therefore very close to locally optimum with respect to T.

8.4 Comparison with Myers' work

Myers did similar performance comparisons of various 1-bit full adder implementations [9], but did not use any optimization. His results, shown in Table 4 below, are quite different from ours, shown in Table 3. Our results show that an appropriate choice of layout and its optimization makes the **Random Logic** circuit better than the **Data Selector**, and that the **PLA** can be made very fast at the expense of Power.

Table 4. 1-bit full adder normalized performance comparison (Myers[9])

type	A	P_{max}	T	APT	PT
Random Logic	100	100	100	100	100
Data Selector	45	50	125	28	72.5
PLA	105	110	170	196	187

8.5. 4-bit Parallel Adder

Tables 1 and 2 show that the locally optimal point of the 1-bit full adder is attained with a pull-down diffusion width of the carry output stage d_{C_0} = 2 or 3, while the corresponding width for the 4-bit parallel adder is d_{C_0} = 8. The pullup width remains 2. This suggests that the critical path passes through the pull-down of the output carry stage, which is indeed the case.

On the hand, for the **Data Selector**, the critical path passes through the pullup of the output carry stage, and in fact it is the pullup width that expands during optimization of the 4-bit parallel adder.

8.6. Comparison of the 1-change and 2-change methods

Figure 8 and Table 5 show a comparison between the 1-change and the 2-change methods when applied to the **Random Logic** implementation. Table 5 is discussed in the next section. The slope of the 2-change method is steeper than that of the 1-change method, but the 2-change method reaches better locally optimal points. Hence in this case the 2-change method works better than the 1-change method does. However, the 2-change method does not work as well as the 1-change method for the **Data Selector**, which has many more parameters. The 2-change method took more iterations than the 1-change method and did not obtain better locally optimal points.

8.7. Effectiveness of our optimization: Cost Improvement ratio

Table 5 below shows the average initial delay times T_0 (obtained from random starts), the average locally optimal delay time T_{opt}, the average percent improvement of the delay time T, and the best locally optimal delay time T_{best}. We can see from this that **2-opt** performs much better than **1-opt**. We should note that it is very important to choose a good order in which to try improvements, because this saves unnecessary search time evaluating changes that are unlikely to be improvements. For example, we chose the diffusion widths of the 3-input NAND gate as the first parameters tried for the **Random Logic** circuit.

Table 5 Cost improvement of our optimization methods

type	opt	criterion	$T_{initial}$	T_{opt}	% improvement	T_{best}
Random Logic	1-opt	T	29.7	19.2	33	19.1
Random Logic	2-opt	T	29.7	16.8	42	16.4
Data Selector	1-opt	T	24.3	17.7	25	15.8
Data Selector	2-opt	T	23.5	18.0	23	15.8
PLA	1-opt	T	19.3	16.3	16	12.8

9. References

[1] P. R. Cappello, K. Steiglitz, "Completely Pipelined Architectures for Digital Signal Processing," *IEEE Trans. on Acoustics, Speech, and Signal Proc.*, vol. ASSP-31, No.4, pp. 1016-22, Aug. 1983.

[2] R. N. Mayo, J. K. Ousterhout, W. S. Scott, "1983 VLSI Tools," Report No. UCB/CSD 83/115, Computer Science Division (EECS), University of California, Berkeley, Calif., March 1983.

[3] S. C. North, "Molding Clay: A Manual for the CLAY Layout Language," VLSI Memo #3, EECS Department, Princeton University, Princeton, N. J., July 1983.

[4] R. J. Lipton, S. C. North, R. Sedgewick, J. Valdes, G. Vijayan, "VLSI Layout as Programming," *ACM Trans. on Programming Languages and Systems*, July 1983.

[5] J. Mata, "ALLENDE User Manual," VLSI Memo #9, EECS Department, Princeton University, Princeton, N. J., May 1984.

[6] R. Rondell, P. C. Treleaven, *VLSI architecture*, Prentice-Hall Inc., Englewood Cliffs, N. J., 1983.

[7] C. Mead, L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co. Menlo Park, Ca., 1980.

[8] J. Mata, "A PLA Generator for the ALLENDE Layout System," EECS Department, Princeton University, Princeton, N. J., June 1984.

[9] D. J. Myers, "Multipliers for LSI and VLSI Signal Processing Applications," Masters Degree thesis, Edinburgh University, Edinburgh, England, Sept. 1981.

[10] R. R. Morita, "Pipelined Architecture for a Cellular Automaton," Senior Independent Project Report, EECS Department, Princeton University, May 1984.

Figure 1. Overall flowchart.

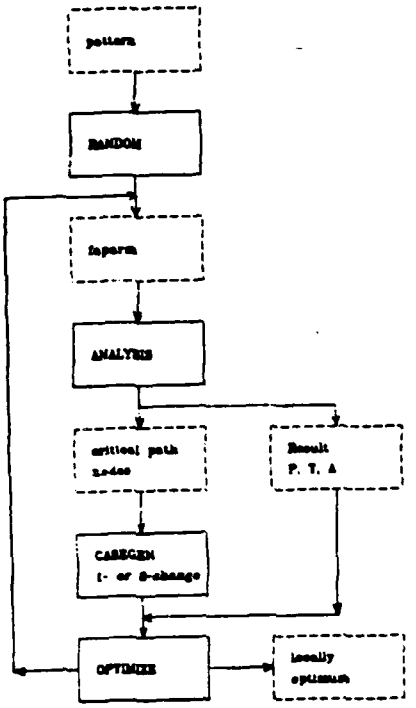


Figure 2. Detail of the ANALYSIS procedure.

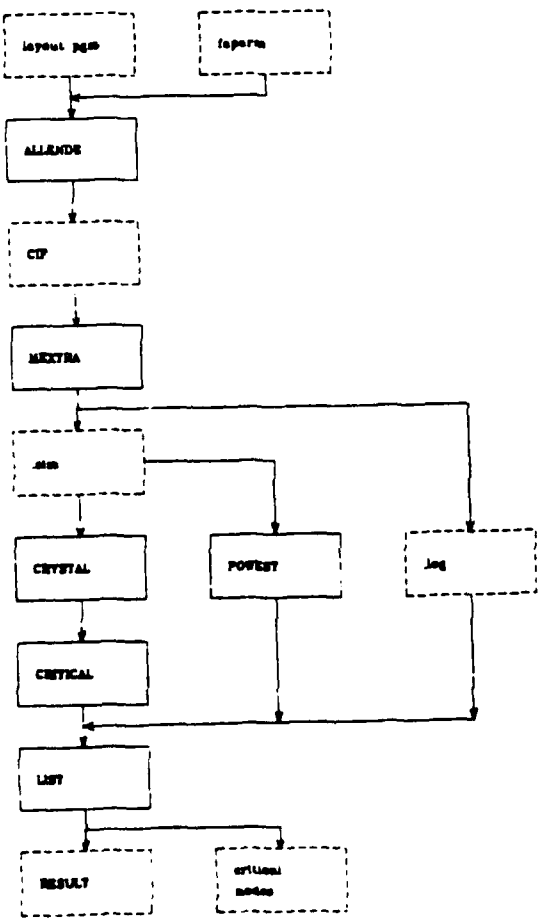
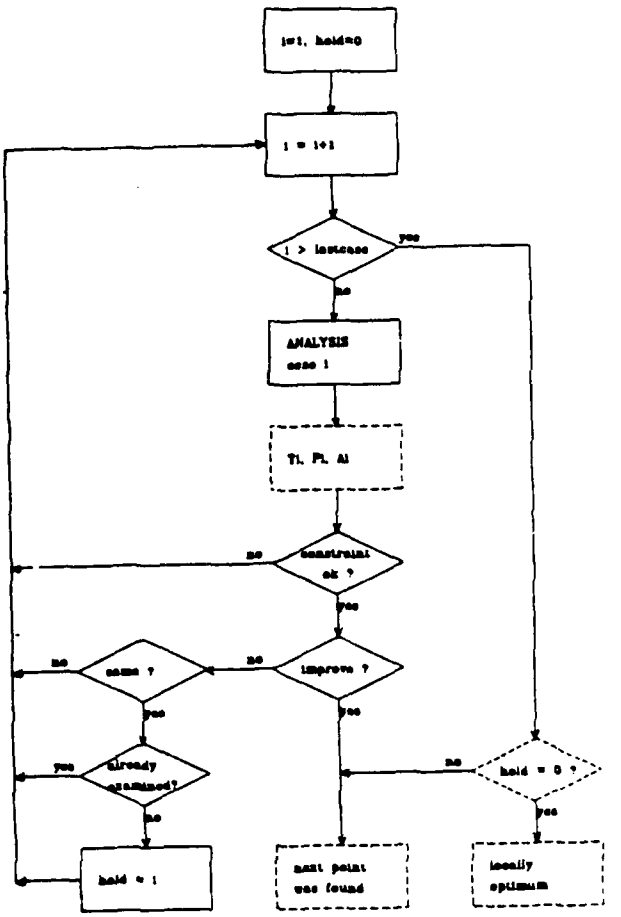


Figure 3. Flowchart of the critical path optimization method.



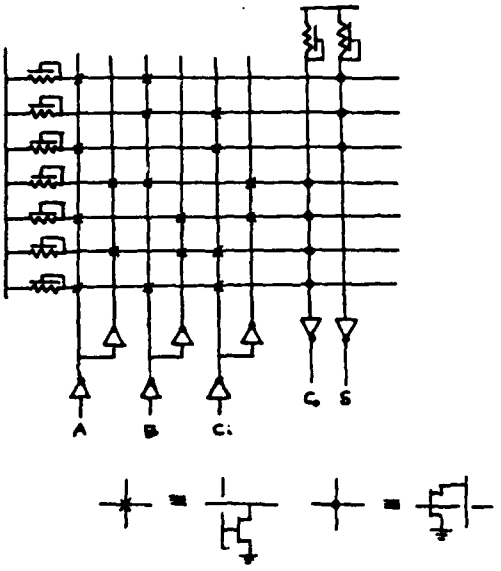


Figure 4. PLA

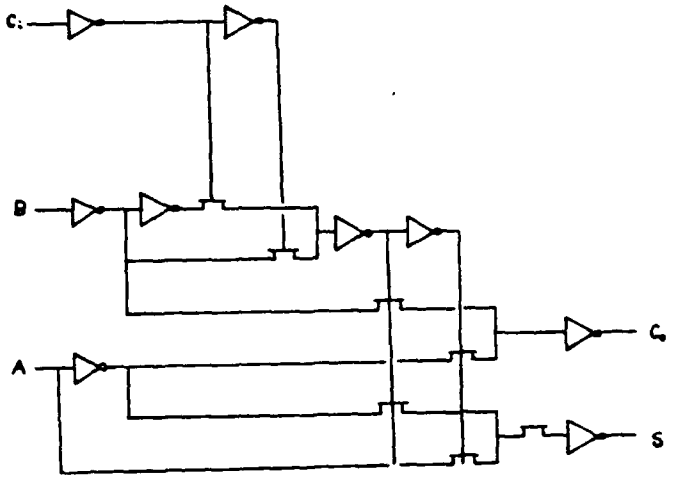


Figure 5. Data Selector

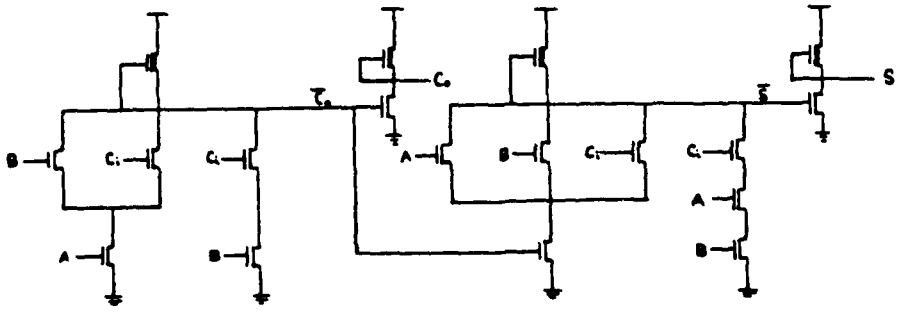


Figure 6. Random Logic

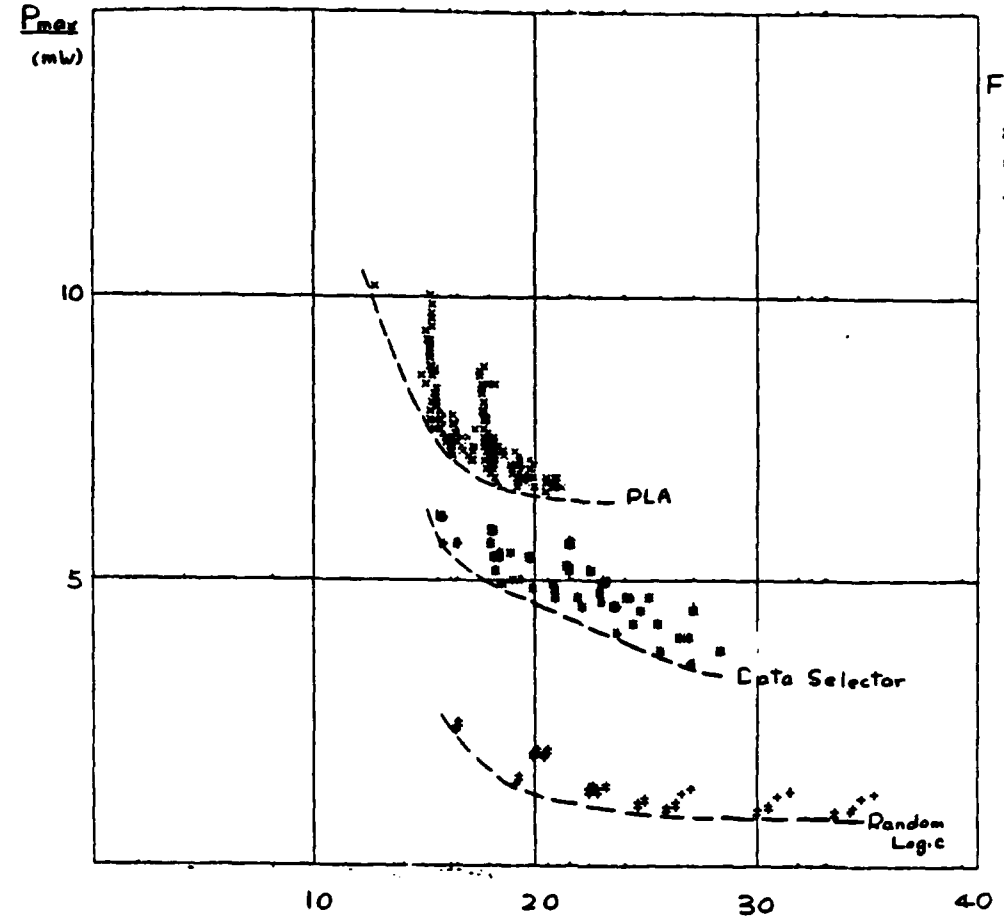


Figure 7

× PLA

* Data Selector

+ Random Logic

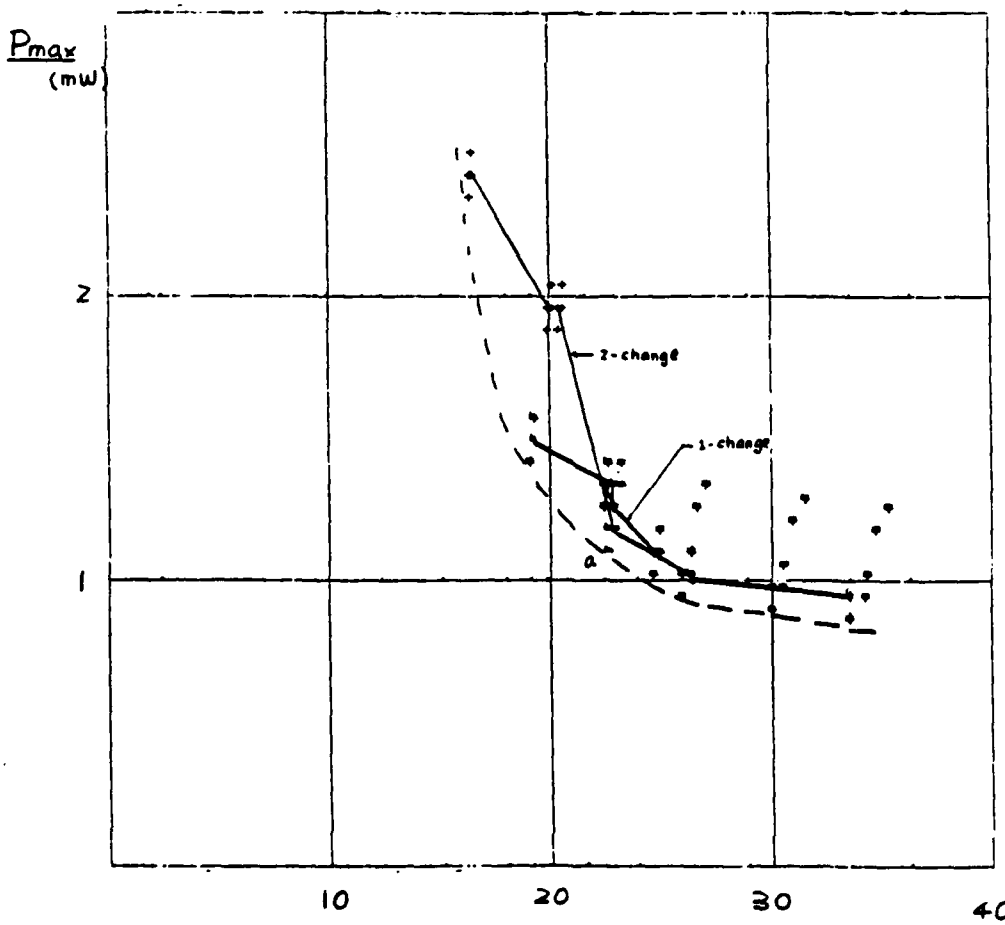


Figure 8

+—+ 2-change

— 1-change

END

FILMED

9-85

DTIC